CS4550

Directed Study

AUTOMATING CONSOLE PORT BASED CONFIGURATON OF CISCO ROUTER WITH PERL SCRIPT

Valter Monteiro Junior
LCDR, Brazilian Navy

September 30, 2002

Prof. Geoffrey Xie

Summer02–Quarter

AUTOMATING CONSOLE PORT BASED CONFIGURATON OF CISCO ROUTER WITH PERL SCRIPT

1. What is the problem?

In this study we define two distinct networks. One is a commercial network, consisting of CISCO routers, switches and workstations; the other is the SAAM (Server and Agent based Active Network Management) network with software agents to optimize allocation of network resources and support user QoS (Quality of Service) requirements.

The objective is to understand those aspects of the Cisco IOS (Internetworking Operating System) that can be configured and/or controlled externally by a SAAM Agent.

2. Why are we solving this problem?

This is part of an on-going effort to find a practical method of uploading the SAAM QoS parameters such as flow definitions and routing table entries into a CISCO router without altering the IOS in the router.

The SAAM network determines Quality of Service parameters based on current network conditions and user requirements. These parameters are dynamic and they must be uploaded into the Cisco routers for them to take effect. It would be desirable to not require any modification to the IOS for such uploads. This is because the IOS is proprietary software. While possible, it is time-consuming and potentially costly to obtain the IOS source code from Cisco or arrange for a special IOS update by Cisco.

3. How else could this problem be solved?

In several previous studies, the focus was on solving this problem using the Simple Network Management Protocol (SNMP), the expected choice for someone who wants to manage a network. But the researchers discovered problems that are best described in Cary Colwell paper [Ref. 7]. After extensive research and experimentation, Cary reported that he did not believe that the pertinent SNMP Management Information Base (MIB) data of a Cisco router are accessible for either reading or writing over the console serial port. With the SNMP approach, we have to do something that is not allowed or expected by the router manufacturer (i.e., Cisco).

4. New Approach

With this observation in mind, a CS4552 student group (spring/2002) changed the approach [Ref. 8]. Instead of working with SNMP, the report of this group suggested using a script language like PERL to open and maintain a TELNET connection with a Cisco router through the console port and utilize the connection to upload the SAAM parameters.

This new approach has the advantage of using a natural mechanism with vendor support to change the routing table and other parameters of the CISCO router configuration. Two issues, however, require further study for this approach to be useful. First, the proposed upload method must have adequate performance (i.e., speed). Second, the method must work for all key SAAM QoS related parameters.

5. Developing our Experience

Before describing the solution in detail, we review some basic concepts and knowledge necessary to this study.

a. CISCO Configuration

This is the main knowledge necessary to design one solution for controlling a Cisco router. The basic configuration steps can be learned from many sources. In the Web Links section, there is enough information from which one can learn to become a good Cisco “programmer”. However, the most effective method to cover this topic is through hands-on experience. It’s very interesting to read Ken Ehresman’s paper – “Configuring CISCO Routers using Telnet” [Ref. 9] — another independent study report that first talks about the Perl module Net::Cisco::Telnet.

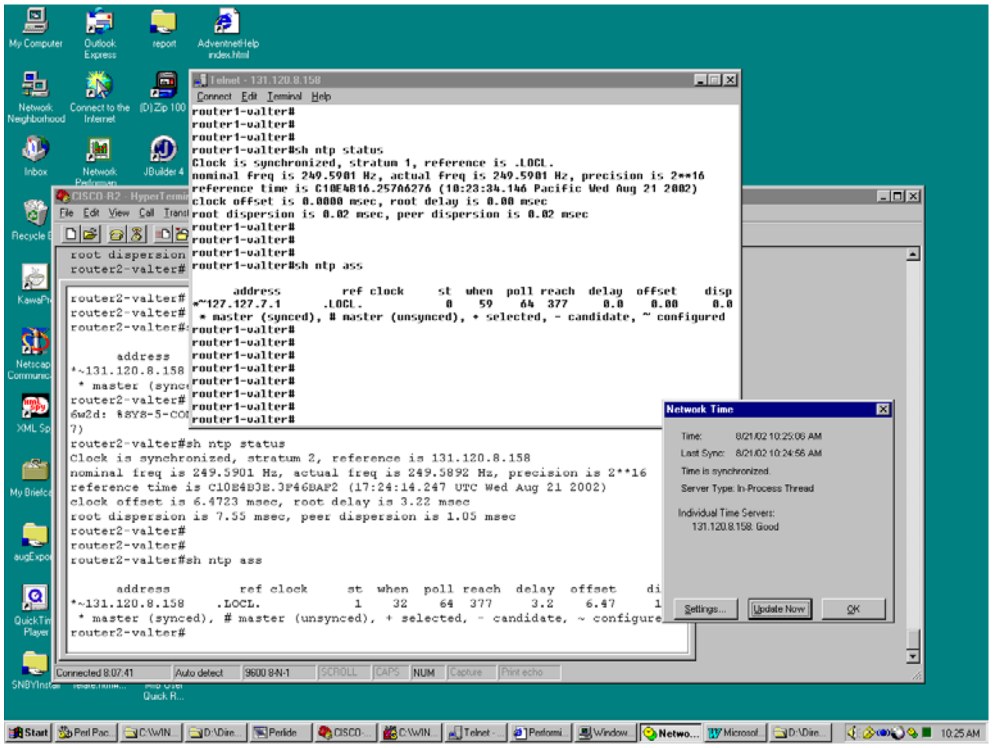
Two Cisco routers were used in this study. They are… The routers were booted with the default configuration that is available in the project archive CD. There is also a TFTP server that you can use to download different start-configuration for the CISCO routers.

b. Clock Synchronization

We configured the CISCO Router-1 to work as NTP (Network Time Protocol) server. We could work with one of the NTP servers available at the NPS network, but since we only needed to record the difference between the start and finish times of commands in a Perl script, a local implementation seemed to be sufficient.

We synchronized the clocks of router-2 and both workstations with the NTP server implemented in router-1.

Below is a screenshot from one of the workstations after time zone configuration and all the devices were synchronized:



Telnet - 131.120.8.158

```
router1-valter#
router1-valter#
router1-valter#
router1-valter#sh ntp status
Clock is synchronized, stratum 1, reference is .LOCL.
nominal freq is 249.5901 Hz, actual freq is 249.5901 Hz, precision is 2**16
reference time is C10E4B16.257A6276 (10:23:34.146 Pacific Wed Aug 21 2002)
clock offset is 0.0000 msec, root delay is 0.00 msec
root dispersion is 0.02 msec, peer dispersion is 0.02 msec
router1-valter#
router1-valter#
router1-valter#
router1-valter#sh ntp ass

      address         ref clock     st  when  poll reach  delay  offset    disp
*~127.127.7.1      .LOCL.            0    59    64  377     0.0    0.00     0.0
 * master (synced), # master (unsynced), + selected, - candidate, ~ configured
router1-valter#
router1-valter#
router1-valter#
router1-valter#
router1-valter#
router1-valter#
router1-valter#
router1-valter#
```

CISCO-R2 - HyperTerminal

```
root dispersion
router2-valter#

router2-valter#
router2-valter#
router2-valter#

      address
*~131.120.8.158
 * master (sync
router2-valter#
6w2d: %SYS-5-CO
7)
router2-valter#sh ntp status
Clock is synchronized, stratum 2, reference is 131.120.8.158
nominal freq is 249.5901 Hz, actual freq is 249.5892 Hz, precision is 2**16
reference time is C10E4B3E.3F46BAF2 (17:24:14.247 UTC Wed Aug 21 2002)
clock offset is 6.4723 msec, root delay is 3.22 msec
root dispersion is 7.55 msec, peer dispersion is 1.05 msec
router2-valter#
router2-valter#
router2-valter#sh ntp ass

      address         ref clock     st  when  poll reach  delay  offset    di
*~131.120.8.158    .LOCL.            1    32    64  377     3.2    6.47     1
 * master (synced), # master (unsynced), + selected, - candidate, ~ configure
router2-valter#
```

Connected 8:07:41 | Auto detect | 9600 8-N-1

Network Time

Time: 8/21/02 10:25:06 AM
Last Sync: 8/21/02 10:24:56 AM
Time is synchronized.
Server Type: In-Process Thread

Individual Time Servers:
131.120.8.158: Good

To work in the Pacific Time zone we had to make additional configurations in the CISCO routers which have a different time zone by default. If we didn’t do that we would work with 2 times zones. Consequently, the PC synchronization with the router would generate a “funny” mistake.

To manually configure the time zone used by the Cisco IOS software, use the following command in global configuration mode:

Command	**Purpose**
Router(config)# **clock timezone** *zone hours-offset* [*minutes-offset*]	Sets the time zone. The *zone* argument is the name of the time zone (typically a standard acronym). The *hours-offset* argument is the number of hours the time zone is different from UTC. The *minutes-offset* argument is the number of minutes the time zone is different from UTC.

The best place to learn about that is the CISCO web page: http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/ffun_c/fcfprt3/fcf012.htm#xtocid28

c. PERL Programming

The development of the required PERL script is the most important part of this study. Definitely as you try to write some code you will learn a lot, but you can do it, as we did, without any kind of prior PERL experience. Of course the code can be incrementally improved. As you learn more you can improve the performance of the code. We learned PERL enough to write the code. The Reference section lists two books that I consulted [Ref. 1, Ref. 2].

It is important to know the basic commands and philosophy of PERL and know how PERL deals with variables. Scalars and lists are crucial concepts too, because, in this particular research, you have to know how to extract state information from the TELNET session.

Other vital information is in the documentation about Net::Telnet and Net::Telnet::Cisco PERL modules. The first one is well described in the chapter 6 of the *Network Programming with PERL* book [Ref. 1]. The second module is so important for our experience that we will talk about it in the next section.

d. Net::Telnet::Cisco Module

The Net::Telnet::Cisco module provides additional functionality to Net::Telnet module for dealing with Cisco routers. Before using this module, it’s necessary to have a good understanding of Net::Telnet. So it is recommended to read *Appendix A* before *Appendix B*.

Recalling the previous research with SNMP, some things are easier to accomplish with SNMP (including SNMP PERL module Net::SNMP). SNMP has three advantages: it’s faster, handles errors better, and doesn’t use any VTYs on the router. SNMP does have some limitations in terms of the types of router parameters that it can modify. In our solution the use of Net::Telnet::Cisco module is critical. Its documentation is included as *Appendix B*.

e. Log File

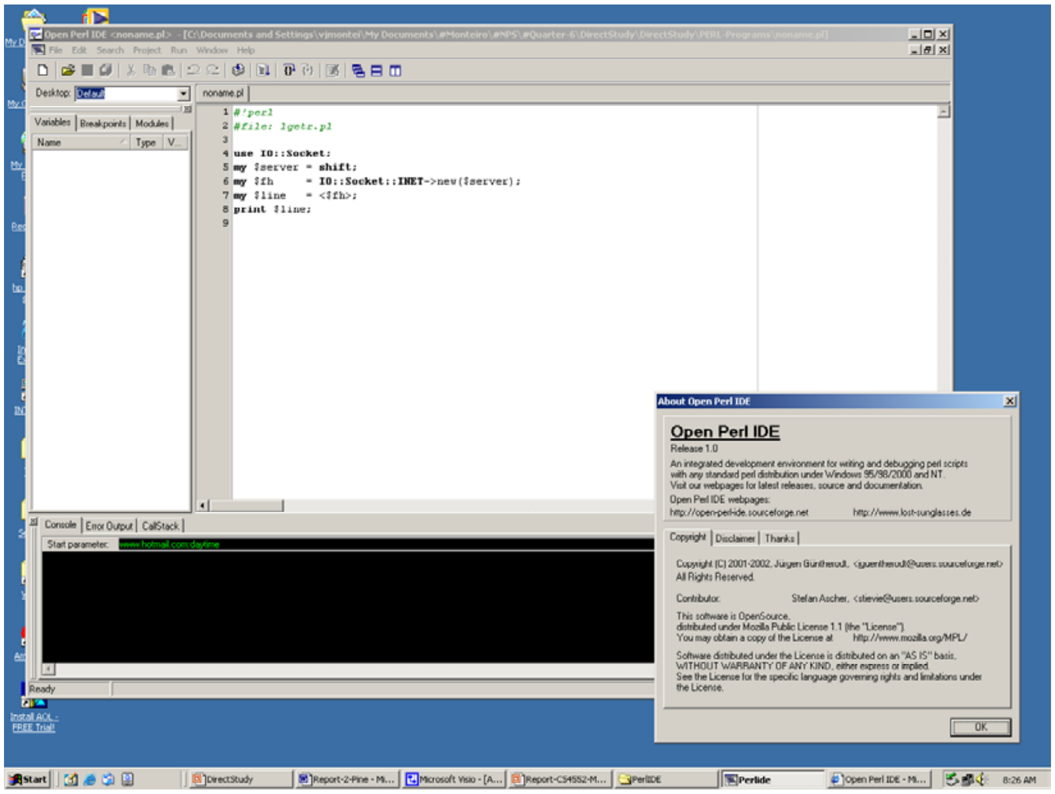
UNIX is the best platform to manage a Log File for the experiment. The syslog system is a standard part of all common UNIX and LINUX distribution. Unfortunately, we are using in this study the Windows NT/2000 log-file functionality. Windows NT/2000 has a similar facility known as the Event Log, but it less straightforward. We found some problems with the Win2K log file, as we could not seem to set the correct timestamp in the Log File even if the host's clock was synchronized win the NTP Server.

f. PERL IDE

Open PERL IDE is an integrated development environment for writing and debugging PERL scripts, which comes with any standard PERL distribution under Windows 95/98/NT/2000.

This software is written in Delphi 5 Object Pascal and PERL and it is Open Source, distributed under Mozilla Public Licence 1.1 and hosted by SourceForge. To know more about this tool, visit its project page to get further information.

Below is a screenshot of the tool. This tool is available in the project archive CD.



```
#!perl
#file: lgetr.pl

use IO::Socket;
my $server = shift;
my $fh     = IO::Socket::INET->new($server);
my $line   = <$fh>;
print $line;
```

Start parameter: www.hotmail.com:daytime

About Open Perl IDE

Open Perl IDE
Release 1.0
An integrated development environment for writing and debugging perl scripts with any standard perl distribution under Windows 95/98/2000 and NT.
Visit our webpages for latest releases, source and documentation.
Open Perl IDE webpages:
http://open-perl-ide.sourceforge.net http://www.lost-sunglasses.de

Copyright (C) 2001-2002, Jürgen Güntherodt, <jguentherodt@users.sourceforge.net>
All Rights Reserved.

Contributor: Stefan Ascher, <stievie@users.sourceforge.net>

This software is OpenSource.
distributed under Mozilla Public License 1.1 (the "License").
You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis,
WITHOUT WARRANTY OF ANY KIND, either express or implied.
See the License for the specific language governing rights and limitations under the License.

Open Perl IDE <noname.pl> - [C:\Documents and Settings\vjmontei\My Documents\#Monteiro\#NPS\#Quarter-6\DirectStudy\DirectStudy\PERL Programs\noname.pl]
File Edit Search Project Run Window Help
Desktop: Default
noname.pl
Variables Breakpoints Modules
Name Type V...
Console Error Output CallStack
Ready
Copyright Disclaimer Thanks
OK
Install AOL - FREE Trial!
Start
DirectStudy
Report-2-Pine - Mi...
Microsoft Visio - [A...
Report-CS4552-M...
PerlIDE
Perlide
Open Perl IDE - Mi...
8:26 AM

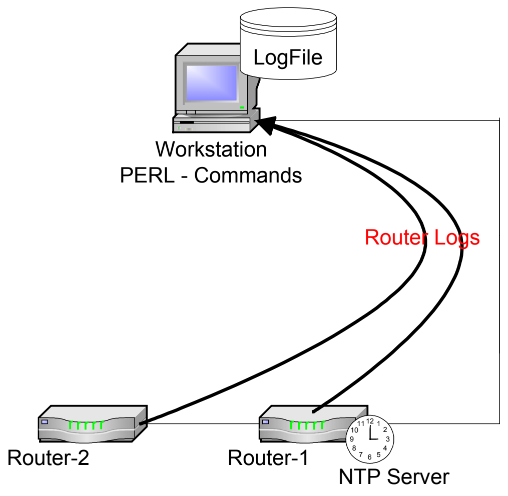
5. Our Experience

a. Scenario

We worked with two routers, whose configurations are described in the with *Appendix C*, and one PC running with Windows NT, where we installed the following applications:

- ActiveState distribution of PERL;
- ActiveState PERL Package Module;
- Kiwi Syslog File;

- NetTime NTP client for synchronization with NTP server installed in CISCO router; and
- Perl script we developed to open TELNET sessions with the routers.



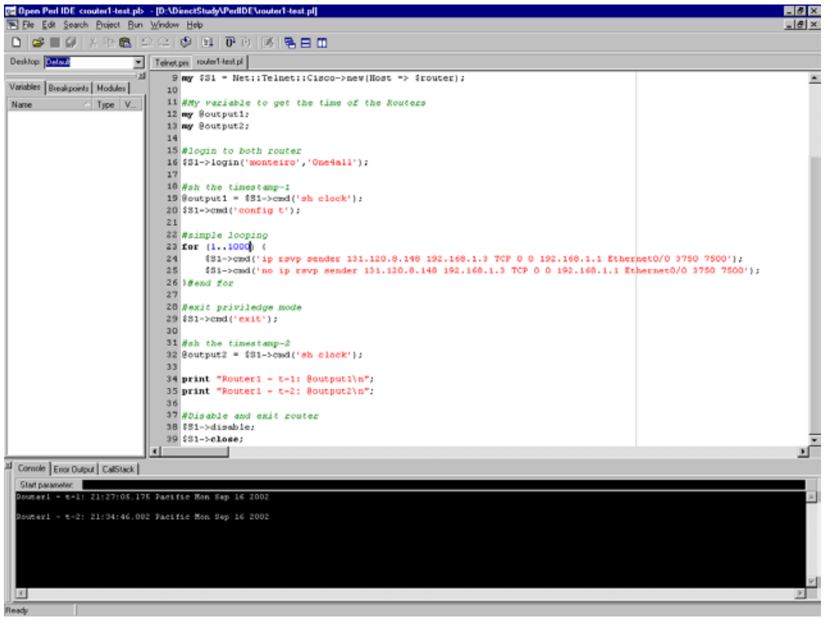
b. Time performance

In order to measure the time that it takes for one single command in PERL script to change the configuration of a CISCO router, we ran a PERL script that was, in a nutshell, a loop with 1000 interactions. Inside this loop we have 2 commands. The first is an *rsvp* CLI command and the other one cancels the action performed by the first command. With this arrangement, we issued a pair of configuration commands to a CISCO router 1000 times.

It's important to note that the Perl script was able to use a single TELNET session with the router to issue all the 2000 commands. This is possible by increasing the timeout value for a TELNET session at the router. See *Appendix D* – Cisco Router Configuration Considerations.

c. Example Measurement

This is the screenshot of the Perl script running 1000 interactions with one router:


Open Perl IDE <router1-test.pl> - [D:\DirectStudy\PerlIDE\router1-test.pl]
File Edit Search Project Run Window Help
Desktop: Default
Variables Breakpoints Modules
Name Type V...
Telnet.pm router1-test.pl
Console Error Output CallStack
Start parameter:
Ready

```
my $S1 = Net::Telnet::Cisco->new(Host => $router);

#My variable to get the time of the Routers
my @output1;
my @output2;

#login to both router
$S1->login('monteiro','One4all');

#sh the timestamp-1
@output1 = $S1->cmd('sh clock');
$S1->cmd('config t');

#simple looping
for (1..1000) {
    $S1->cmd('ip rsvp sender 131.120.8.148 192.168.1.3 TCP 0 0 192.168.1.1 Ethernet0/0 3750 7500');
    $S1->cmd('no ip rsvp sender 131.120.8.148 192.168.1.3 TCP 0 0 192.168.1.1 Ethernet0/0 3750 7500');
}#end for

#exit priviledge mode
$S1->cmd('exit');

#sh the timestamp-2
@output2 = $S1->cmd('sh clock');

print "Router1 - t-1: @output1\n";
print "Router1 - t-2: @output2\n";

#Disable and exit router
$S1->disable;
$S1->close;
```

```
Router1 - t-1: 21:27:05.175 Pacific Mon Sep 16 2002

Router1 - t-2: 21:34:46.002 Pacific Mon Sep 16 2002
```

The bottom part of the screenshot with black background shows time measurements for the interactions:

T1 = 21:27:05.175

T2 = 21:34:46.082

T1 – T2 = 7 min 40.907 sec = 460.907 sec

(T1-T2)/2000 = 230.253 ms to execute each command.

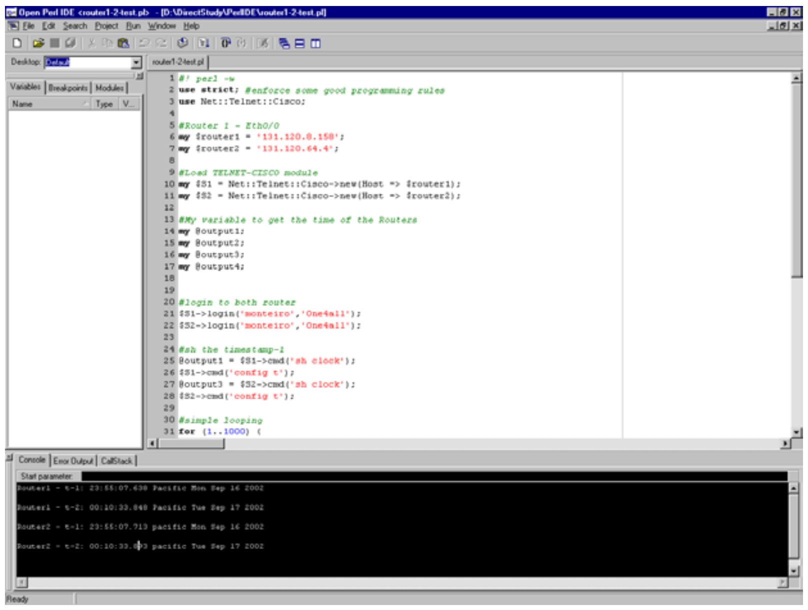
6. Experiment with 2 Routers

This time we worked with 2 routers. We opened 2 TELNET sessions, one in each router.

In this try we could conclude that the time spent to change the CISCO router configuration when we work with one router is almost the same as when you work with 2 routers. We can expand our conclusion and estimate the time to work with 100 CISCO routers.

a. Time Measurement

This is the screenshot of the Perl script conducting 1000 interactions with two routers:


Open Perl IDE <router1-2-test.pl> - [D:\DirectStudy\PerlIDE\router1-2-test.pl]
File Edit Search Project Run Window Help
Desktop: Default
router1-2-test.pl
Variables Breakpoints Modules
Name Type V...
Console Error Output CallStack
Start parameter:
Ready

```
#! perl -w
use strict; #enforce some good programming rules
use Net::Telnet::Cisco;

#Router 1 - Eth0/0
my $router1 = '131.120.8.158';
my $router2 = '131.120.64.4';

#Load TELNET-CISCO module
my $S1 = Net::Telnet::Cisco->new(Host => $router1);
my $S2 = Net::Telnet::Cisco->new(Host => $router2);

#My variable to get the time of the Routers
my @output1;
my @output2;
my @output3;
my @output4;


#login to both router
$S1->login('monteiro','One4all');
$S2->login('monteiro','One4all');

#sh the timestamp-1
@output1 = $S1->cmd('sh clock');
$S1->cmd('config t');
@output3 = $S2->cmd('sh clock');
$S2->cmd('config t');

#simple looping
for (1..1000) {
```

```
Router1 - t-1: 23:55:07.638 Pacific Mon Sep 16 2002

Router1 - t-2: 00:10:33.848 Pacific Tue Sep 17 2002

Router2 - t-1: 23:55:07.713 pacific Mon Sep 16 2002

Router2 - t-2: 00:10:33.893 pacific Tue Sep 17 2002
```

The new timestamps are:

Router-1 t1: 23:55:07.638

Router-1 t2: 00:10:33.848

Router-2: t1: 23:55:07.713

Router-2 t2: 00:10:33.893

The time between the T1 of the router 1 and T2 of the router 2 show us:

T1 – T2 = 15.23758 min = 926.255 sec

(T1-T2)/4000 = 230.156 ms to execute one command in a router, working with two routers simultaneously.

Therefore, the difference between the first result with one router and the second result with two routers is minimal:

1^{o} $\text{result}_{(1 \text{ router})}$ – 2^{o} $\text{result}_{(2 \text{ routers})}$ = 230.253 ms - 230.156 ms = 0.131 ms

We can estimate that for 100 routers, the script would spend about 13.1 ms more to operate all of them simultaneously than operating one router separately 100 times.

7. Conclusion

This study strongly suggests the feasibility of using of PERL script as a tool to change a CISCO router’s configuration such as adding an entry to its routing table.

This study is a big step to towards the main goal that is to interface SAAM QoS network parameters with a commercial network. The SNMP approach, very well studied in previous researches, seems to be less promising in comparison.

The average time of 0.230 seconds to run a command in a CISCO router, and the fact that this time doesn’t have a significant increase when we work with many routers is, undoubtedly, a very useful result that warrants further research and implementations.

8. References

1. Network Programming with PERL – Lincoln D. Stein
2. Learning PERL – O’Reilly
3. Net::TELNET PERL Module – Appendix A

4. Net::TELNET::Cisco PERL module – Appendix B
5. CCNA – Cisco Press; Wendell Odom
6. Configuring CISCO Routers using Telnet, Key Ehresman
7. Experiment with CISCO MIBs, QoS, and SNMP Features for Adapting SAAM Flow/Path Based Routing and Control, Cary Colwell
8. Programming CISCO Router using SNMP, Spring/2002 CS4552 Project Report – Kyriakos Sergis, Michael Hadley and Valter Monteiro Jr.

9. Web Links

Icon	URL	Item of interesting
	http://sourceforge.net/projects/nettime/	Synchronization
	http://www.cisco.com	Cisco web page
	http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/fun_r/frprt3/frd3003.htm#xtocid2298129	Basic System Management Commands – CISCO NTP
	http://www.perldoc.com/	PERL 5.8.0 Documentation
	http://www.perl.org/	PERL news
Net::Telnet::Cisco	http://nettelnetcisco.sourceforge.net/	Net::Telnet::Cisco PERL module main page with Documentation, Download, etc..

	http://www.cpan.org/	**Comprehensive PERL Archive Network**
	http://open-PERL-ide.sourceforge.net/	Open PERL IDE is an integrated development environment for writing and debugging PERL scripts with any standard PERL distribution under Windows 95/98/NT/2000.

10. Learning Side Effects

The main learning side effects of this Direct Study were improvement in CISCO router configuration, RSVP understanding, PERL learning, Log Files configurations, System Synchronization, NTP Server configuration, TCP/IP protocol implementation, strengths and weaknesses of TCP/IP protocols including TELNET, NTP, and FTP.

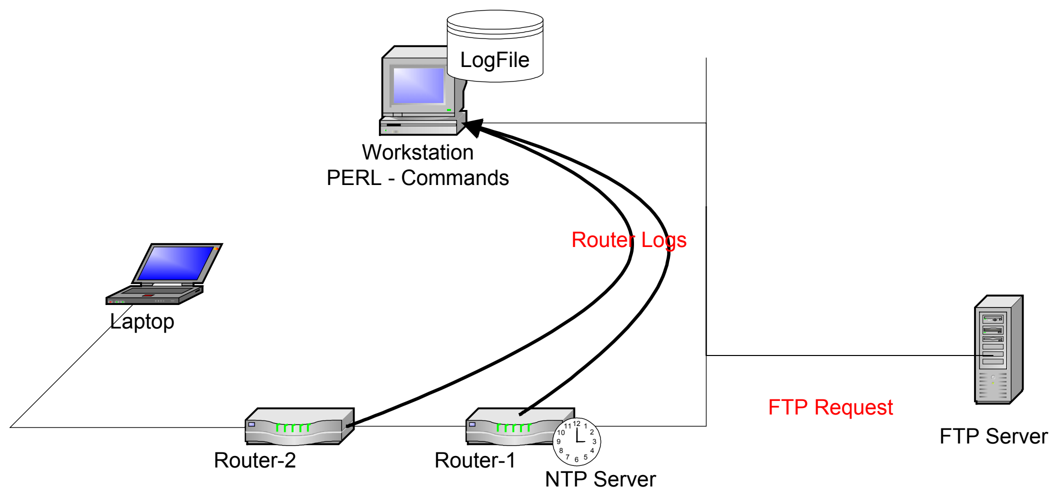
We also learned some “practical tricks” of network configuration, implementation, measurement and performance proof.

11. Future Work

Implement application flows

We started this experiment, but, unfortunately, we didn’t have enough time to conclude it. It requires 2 PCs and 1 FTP server. The FTP server is plugged “behind” the routers, so in order to open a FTP session the packets have to go through both routers. (See picture below.) The first PC opens 2 FTP sessions. After that the second PC opens a third FTP session. A PERL script has been run to perform bandwidth reservation for the third FTP session. One can watch in the Log File that the last FTP session proceeded faster than the others.

The result will show that we can effectively change the configuration of a network with PERL scripts to create a flow path with specific QoS guarantees for an application.



Repeat same experiments using the LINUX platform

This is very important because Linux is a more suitable platform for this type of experiments. Using WinNT, we had some difficulty in time synchronization and dealing with the

Log File. After all LINUX is open source and it would be possible to modify its behavior as required by the experiments. Another advantage is that PERL seems to work better with the LINUX platform.

Experiment with different Cisco commands

We only did some tests with just a couple of RSVP commands in a superficial network. We need to experiment with all the Cisco commands that will be potentially used by the SAAM network. We also need to measure the performance of PERL scripts and their capability to change the CISCO router configuration in a more realistic network.

Integrate PERL with Java

The Perl scripts need to be integrated into the SAAM system. As the current SAAM test-bed network were developed in Java, we need to explore ways to integrate PERL scripts with a Java program.

There is one application that may help this integration. JPL – Java/PERL Lingo integration kit is a part of the commercial release of the PERL software contained in the “PERL Resource Kit for UNIX” by “O’Reilly Associates”. JPL was written from scratch by Larry Wall (the creator of PERL) to allow for the writing of PERL/Java hybrid application. See in the Link session the URL that contains more information about this product.

There is also a windows resource kit version, but this version doesn’t come with JPL.

Implement Security – A mandatory future research that we have to implement in order to have a real product is the security of this application. We have to develop a Secure SHell (SSH) application instead of TELNET application. This is not difficult as this capability is available in CISCO router and there is support of it in PERL. To do this, the main sources of information are the chapter 6 of the Networking Programming with PERL book [Ref. 1] and the CISCO paper - Configuring Secure Shell on CISCO Routers, which is in the project archive CD.

Appendix A

Net::Telnet PERL Module Documentation

As this documentation has more than 30 pages, in order to save some trees, we prefer to indicate the URL where you can find it: http://theoryx5.uwinnipeg.ca/CPAN/data/Net-Telnet/Net/Telnet.html

Appendix B

Net::Telnet::Cisco PERL Module Documentation

NAME

Net::Telnet::Cisco - interact with a Cisco router

SYNOPSIS

```
  use Net::Telnet::Cisco;
  my $session = Net::Telnet::Cisco->new(Host =>
'123.123.123.123');
  $session->login('login', 'password');
  # Execute a command
  my @output = $session->cmd('show version');
  print @output;
  # Enable mode
  if ($session->enable("enable_password") ) {
      @output = $session->cmd('show privilege');
      print "My privileges: @output\n";
  } else {
      warn "Can't enable: " . $session->errmsg;
  }
  $session->close;
```

DESCRIPTION

Net::Telnet::Cisco provides additional functionality to Net::Telnet for dealing with Cisco routers.

`cmd()` parses router-generated error messages - the kind that begin with a '%' - and stows them in $obj->errmsg(), so that errmode can be used to perform automatic error-handling actions.

CAVEATS

Before you use Net::Telnet::Cisco, you should have a good understanding of Net::Telnet, so read it's documentation first, and then come back here to see the improvements.

Some things are easier to accomplish with UCD's C-based SNMP module, or the all-perl Net::SNMP. SNMP has three advantages: it's faster, handles errors better, and doesn't use any VTYs on the router. SNMP does have some limitations, so for anything you can't accomplish with SNMP, there's Net::Telnet::Cisco.

METHODS

new - create new Net::Telnet::Cisco object

```
    $session = Net::Telnet::Cisco->new(
        [Autopage                  => $boolean,] # 1
        [More_prompt                         =>  $matchop,]  #
'/(?m:^\s*--More--)/',
        [Always_waitfor_prompt    => $boolean,] # 1
        [Waitfor_pause            => $milliseconds,] # 0.1
        [Normalize_cmd            => $boolean,] # 1
        [Send_wakeup              => $when,] # 0
        [Ignore_warnings          => $boolean,] # 0
        [Warnings                 => $matchop,] # see docs

        # Net::Telnet arguments
        [Binmode                  => $mode,]
        [Cmd_remove_mode          => $mode,]
        [Dump_Log                 => $filename,]
        [Errmode                  => $errmode,]
        [Fhopen                    => $filehandle,]
        [Host                      => $host,]
        [Input_log                 => $file,]
        [Input_record_separator    => $char,]
        [Option_log                => $file,]
        [Output_log                => $file,]
        [Output_record_separator   => $char,]
        [Port                      => $port,]
        [Prompt                    => $matchop,] # see docs
        [Telnetmode                => $mode,]
        [Timeout                   => $secs,]
    );
```

Creates a new object. Read `perldoc perlboot` if you don't understand that.

login - login to a router

```
    $ok = $obj->login($username, $password);
    $ok = $obj->login([Name     => $username,]
                      [Password => $password,]
                      [Passcode => $passcode,] # for Secur-
ID/XTACACS
                      [Prompt   => $match,]
                      [Timeout  => $secs,]);
```

All arguments are optional as of v1.05. Some routers don't ask for a username, they start the login conversation with a password request.

cmd - send a command

```
    $ok = $obj->cmd($string);
    $ok = $obj->cmd(String   => $string,
                    [Output  => $ref,]
                    [Prompt  => $match,]
                    [Timeout => $secs,]
                    [Cmd_remove_mode => $mode,]);
    @output = $obj->cmd($string);
    @output = $obj->cmd(String   => $string,
                        [Output  => $ref,]
                        [Prompt  => $match,]
                        [Timeout => $secs,]
                        [Cmd_remove_mode => $mode,]
                        [Normalize_cmd => $boolean,]);
```

Normalize_cmd has been added to the default Net::Telnet args. It lets you temporarily change whether backspace, delete, and kill characters are parsed in the command output. (This is performed by default)

prompt - return control to the program whenever this string occurs in router output

```
    $matchop = $obj->prompt;
    $prev = $obj->prompt($matchop);
```

The default cmd_prompt changed in v1.05. It's suitable for matching prompts like `router$` , `router#` , `router> (enable)` , and `router(config-if)#`

Let's take a closer look, shall we?

```
  (?m:                  # Net::Telnet doesn't accept quoted
regexen (i.e. qr//)
                        # so we need to use an embedded
pattern-match modifier
                        # to treat the input as a multiline
buffer.
    ^                   # beginning of line
      [\w.-]+           # router hostname
      \s?               # optional space
      (?:                    # Strings like "(config)" and
"(config-if)", "(config-line)",
                        # and "(config-router)" indicate
that we're in privileged
```

```
        \(config[^\)]*\) # EXEC mode (i.e. we're enabled).
      )?               # The middle backslash is only there
to appear my syntax
                       # highlighter.
      \s?              # more optional space
      [\$#>]             # Prompts typically end with "$",
"#", or ">". Backslash
                       # for syntax-highlighter.
      \s?              # more space padding
      (?:              # Catalyst switches print "(enable)"
when in privileged
        \(enable\)     # EXEC mode.
      )?
      \s*                  # spaces before the end-of-line
aren't important to us.
    $                  # end of line
  )                    # end of (?m:
```

The default prompt published in 1.03 was `/^\s*[\w().-]*[\$#>]\s?(?:\(enable\))?\s*$/`. As you can see, the prompt was drastically overhauled in 1.05. If your code suddenly starts timing out after upgrading Net::Telnet::Cisco, this is the first thing to investigate.

enable - enter enabled mode

```
    $ok = $obj->enable;
    $ok = $obj->enable($password);
    $ok = $obj->enable([Name => $name,] [Password =>
$password,]
                       [Passcode => $passcode,] [Level =>
$level,]);
```

This method changes privilege level to enabled mode, (i.e. root)

If a single argument is provided by the caller, it will be used as a password. For more control, including the ability to set the privilege-level, you must use the named-argument scheme.

`enable()` returns 1 on success and undef on failure.

is_enabled - Am I root?

```
    $bool = $obj->is_enabled;
```

A trivial check to see whether we have a root-style prompt, with either the word ``(enable)'' in it, or a trailing ``#''.

Warning: this method will return false positives if your prompt has ``#''s in it. You may be better off calling `$obj->cmd("show privilege")` instead.

disable - leave enabled mode

```
    $ok = $obj->disable;
```

This method exits the router's privileged mode.

ios_break - send a break (control-^)

```
    $ok = $obj->ios_break;
```

You may have to use errmode(), fork, or threads to break at the an appropriate time.

last_prompt - displays the last prompt matched by `prompt()`

```
    $match = $obj->last_prompt;
```

`last_prompt()` will return '' if the program has not yet matched a prompt.

always_waitfor_prompt - waitfor and cmd prompt behaviour

```
    $boolean = $obj->always_waitfor_prompt;
    $boolean = $obj->always_waitfor_prompt($boolean);
```

Default value: 1

If you pass a Prompt argument to `cmd()` or `waitfor()` a String or Match, they will return control on a successful match of your `argument(s)` or the default prompt. Set always_waitfor_prompt to 0 to return control only for your arguments.

This method has no effect on login(). login() will always wait for a prompt.

waitfor_pause - insert a small delay before waitfor()

```
    $boolean = $obj->waitfor_pause;
    $boolean = $obj->waitfor_pause($milliseconds);
```

Default value: 0.1

In rare circumstances, the last_prompt is set incorrectly. By adding a very small delay before calling the parent class's waitfor(), this bug is eliminated. If you ever find reason to modify this from it's default setting, please let me know.

autopage - Turn autopaging on and off

```
    $boolean = $obj->autopage;
    $boolean = $obj->autopage($boolean);
```

Default value: 1

IOS pages output by default. It expects human eyes to be reading the output, not programs. Humans hit the spacebar to scroll page by page so autopage() mimicks that behaviour. This is the slow way to handle paging. See the Paging EXAMPLE for a faster way.

normalize_cmd - Turn normalization on and off

```
    $boolean = $obj->normalize_cmd;
    $boolean = $obj->normalize_cmd($boolean);
```

Default value: 1

IOS clears '--More--' prompts with backspaces (e.g. ^H). If you're excited by the thought of having raw control characters like ^H (backspace), ^? (delete), and ^U (kill) in your command output, turn this feature off.

Logging is unaffected by this setting.

more_prompt - Matchop used by autopage()

```
    $matchop = $obj->prompt;
    $prev = $obj->prompt($matchop);
```

Default value: '/(?m:\s*--More--)/'.

Please email me if you find others.

send_wakeup - send a newline to the router at login time

```
    $when = $obj->send_wakeup;
    $when = $obj->send_wakeup( 'connect' );
    $when = $obj->send_wakeup( 'timeout' );
    $when = $obj->send_wakeup( 0 );
```

Default value: 0

Some routers quietly allow you to connect but don't display the expected login prompts. Sends a newline in the hopes that this spurs the routers to print something.

'connect' sends a newline immediately upon connection. 'timeout' sends a newline if the connection timeouts. 0 turns this feature off.

I understand this works with Livingston Portmasters.

ignore_warnings - Don't call error() for warnings

```
    $boolean = $obj->ignore_warnings;
    $boolean = $obj->ignore_warnings($boolean);
```

Default value: 0

Not all strings that begin with a '%' are really errors. Some are just warnings. By setting this, you are ignoring them. This will show up in the logs, but that's it.

warnings - Matchop used by ignore_warnings().

```
    $boolean = $obj->warnings;
    $boolean = $obj->warnings($matchop);
```

Default value:

```
        /(?mx:^% Unknown VPN
             |^%IP routing table VRF.* does not exist. Create first$
             |^%No CEF interface information
             |^%No matching route to delete$
             |^%Not all config may be removed and may
reappear after reactivating
         )/
```

Not all strings that begin with a '%' are really errors. Some are just warnings. Cisco calls these the CIPMIOSWarningExpressions.

EXAMPLES

Paging

v1.08 added internal autopaging support to cmd(). Whenever a '--Page--' prompt appears on the screen, we send a space right back. It works, but it's slow. You'd be better off sending one of the following commands just after login():

```
  # To a router
  $session->cmd('terminal length 0');
  # To a switch
  $session->cmd('set length 0');
```

Logging

Want to see the session transcript? Just call input_log().

```
  e.g.
  my $session = Net::Telnet::Cisco->new(Host => $router,
                                        Input_log        =>
"input.log",
                                        );
```

See input_log() in the Net::Telnet manpage for info.

Input logs are easy-to-read translated transcripts with all of the control characters and telnet escapes cleaned up. If you want to view the raw session, see dump_log() in the Net::Telnet manpage. If you're getting tricky and using print() in addition to cmd(), you may also want to use output_log().

Big output

Trying to dump the entire BGP table? (e.g. ``show ip bgp'') The default buffer size is 1MB, so you'll have to increase it.

```
  my $MB = 1024 * 1024;
  $session->max_buffer_length(5 * $MB);
```

Sending multiple lines at once

Some commands like ``extended ping'' and ``copy'' prompt for several lines of data. It's not necessary to change the prompt for each line. Instead, send everything at once, separated by newlines.

For:

```
  router# ping
  Protocol [ip]:
  Target IP address: 10.0.0.1
  Repeat count [5]: 10
  Datagram size [100]: 1500
  Timeout in seconds [2]:
  Extended commands [n]:
  Sweep range of sizes [n]:
```

Try this:

```
  my $protocol  = ''; # default value
  my $ip       = '10.0.0.1';
  my $repeat    = 10;
  my $datagram  = 1500;
  my $timeout   = ''; # default value
  my $extended  = ''; # default value
  my $sweep     = ''; # default value
  $session->cmd(
  "ping
  $protocol
  $ip
  $repeat
  $datagram
  $timeout
  $extended
  $sweep
  ");
```

If you prefer, you can put the cmd on a single line and replace every static newline with the ``\n'' character.

e.g.

```
  $session->cmd("ping\n$protocol\n$ip\n$repeat\n$datagram\n"
              . "$timeout\n$extended\n$sweep\n");
```

Backup via TFTP

Backs up the running-confg to a TFTP server. Backup file is in the form ``router-confg''. Make sure that file exists on the TFTP server or the transfer will fail!

```
  my $backup_host  = "tftpserver.somewhere.net";
  my $device       = "cisco.somewhere.net";
  my $type         = "router"; # or "switch";
  my $ios_version  = 12;
  my @out;
  if ($type eq "router") {
      if ($ios_version >= 12) {
          @out = $session->cmd("copy system:/running-config
"
                              .      "tftp://$backup_host/$device-
confg\n\n\n");
      } elsif ($ios_version >= 11) {
          @out    =    $session->cmd("copy    running-config
tftp\n$backup_host\n"
                              . "$device-confg\n");
      } elsif ($ios_version >= 10) {
          @out            =            $session->cmd("write
net\n$backup_host\n$device-confg\n\n");
      }
  } elsif ($type eq "switch") {
      @out = $session->cmd("copy system:/running-config "
                    .          "tftp://$backup_host/$device-
confg\n\n\n");
  }
```

SUPPORT

http://NetTelnetCisco.sourceforge.net/

Mailing lists

nettelnetcisco-announce is for important security bulletins and upgrades. Very low traffic, no spam, **HIGHLY RECOMMENDED!**
http://lists.sourceforge.net/lists/listinfo/nettelnetcisco-announce

nettelnetcisco-users is for usage discussion, help, tips, tricks, etc.
http://lists.sourceforge.net/lists/listinfo/nettelnetcisco-users

nettelnetcisco-devel is for uber-hackers; you know who you are.
http://lists.sourceforge.net/lists/listinfo/nettelnetcisco-devel

Help/discussion forums

http://sourceforge.net/forum/?group_id=48856

Bug tracker

http://sourceforge.net/tracker/?group_id=48856

SEE ALSO

the Net::Telnet manpage

the Net::SNMP manpage

UCD NetSNMP - http://www.netsnmp.org/

RAT/NCAT - http://ncat.sourceforge.net/

AUTHOR

Joshua_Keroes@eli.net $Date: 2002/06/18 17:17:03 $

It would greatly amuse the author if you would send email to him and tell him how you are using Net::Telnet::Cisco.

As of Mar 2002, 170 people have emailed me. N::T::C is used to help manage over 14,000 machines! Keep the email rolling in!

Appendix C

Router-1 configuration:

```
router1#
Using 1658 out of 29688 bytes
!
! Last configuration change at 22:18:03 Pacific Wed Sep 18
2002 by monteiro
! NVRAM config last updated at 22:18:24 Pacific Wed Sep 18
2002 by monteiro
!
version 12.0
service timestamps debug datetime msec localtime show-
timezone
service timestamps log datetime msec localtime show-timezone
service password-encryption
!
hostname router1
!
boot system flash:c2600-d-mz.120-7.T.bin
logging buffered 10000 debugging
no logging console
enable secret 5 $1$w8SY$/GDaRfBcSEr8F6tbIAKUP1
!
username cs4552 password 7 020116541E165B
username monteiro privilege 15 password 7 091C400C4D041B1E
username xie privilege 15 password 7 15420509502B2728
!
!
!
!
memory-size iomem 10
clock timezone Pacific -7
ip subnet-zero
ip domain-name cs.nps.navy.mil
ip name-server 131.120.18.40
ip name-server 131.120.18.41
!
!
!
!
interface Ethernet0/0
 ip address 131.120.8.158 255.255.252.0
 no ip directed-broadcast
 fair-queue 1200 256 234
 ip rsvp bandwidth 7500 7500
!
interface Ethernet0/1
 ip address 131.120.64.2 255.255.255.224
 ip directed-broadcast
 full-duplex
 fair-queue 1200 256 234
 ip rsvp bandwidth 7500 7500
!
router rip
 version 2
 redistribute static
 network 131.120.0.0
!
ip classless
ip route 0.0.0.0 0.0.0.0 131.120.8.1
no ip http server
ip   rsvp   sender   192.168.1.2   131.120.8.137   TCP   0   0
131.120.8.158 Ethernet0/0 7500
7500
ip  rsvp  reservation  192.168.1.2  131.120.8.137  TCP  0  0
131.120.8.158 Ethernet0/0
FF LOAD 7500 7500
!
logging trap debugging
logging 131.120.8.137
access-list 1 permit any log
!
line con 0
 transport input none
line aux 0
line vty 0 4
 exec-timeout 50 0
 password 7 104D1A4D504240
 login local
!
ntp master 1
no scheduler allocate
end

router1#
```

Router-2 configuration:

```
router2#
Using 1612 out of 29688 bytes
!
! Last configuration change at 22:55:55 pacific Wed Sep 18
2002 by monteiro
! NVRAM config last updated at 22:56:05 pacific Wed Sep 18
2002 by monteiro
!
version 12.0
service timestamps debug datetime msec localtime show-
timezone
service timestamps log uptime
service password-encryption
!
hostname router2
!
logging buffered 10000 debugging
no logging console
enable password 7 135519175F0D0826
!
username monteiro privilege 15 password 7 02560A5E5F07032D
username xie privilege 15 password 7 15420509502B272868
!
!
!
!
memory-size iomem 10
clock timezone pacific -7
ip subnet-zero
no ip ftp passive
ip domain-name cs.nps.navy.mil
ip name-server 131.120.154.52
ip name-server 131.120.18.40
!
!
!
!
interface Ethernet0/0
 description connection to 525 servers
 ip address 192.168.1.1 255.255.255.0
 no ip directed-broadcast
 full-duplex
 fair-queue 1200 256 234
 ip rsvp bandwidth 7500 7500
!
```

```
interface Ethernet0/1
 ip address 131.120.64.4 255.255.255.224
 no ip directed-broadcast
 full-duplex
 fair-queue 1200 256 234
 ip rsvp bandwidth 7500 7500
!
router rip
 version 2
 network 131.120.0.0
 network 192.168.1.0
!
ip default-gateway 131.120.64.2
ip classless
no ip http server
ip   rsvp   sender   192.168.1.2   131.120.8.137   TCP   0   0
131.120.8.158 Ethernet0/0 7500
7500
ip  rsvp  reservation  192.168.1.2  131.120.8.137  TCP  0  0
192.168.1.1 Ethernet0/0 FF
 LOAD 7500 7500
!
logging 131.120.8.137
access-list 100 permit tcp any any
!
line con 0
 login
 transport input none
line aux 0
line vty 0 4
 exec-timeout 50 0
 password 7 130604465E5956
 login local
!
ntp clock-period 17208487
ntp server 131.120.8.158 prefer
end

router2#
```

Appendix D

Cisco Router Configuration Considerations

To best repeat this experiment we have to make some special configurations in CISCO router:

- **Implement super-users** – This will help because it’s not necessary to enter in the privilege mode after the login. However, this command is a security issue and one has to deal with this trade-off.

  Commands:

  ```
  Router-1# username xie pri 15 pass professor.
  ```

- **Increase the TELNET session time out** – This will help because it ensures that a TELNET session can be used to transport Cisco commands to a router for an extended period of time even if there may be substantial idle time between two commands. However, this brings up another security issue. The best approach may be the following. In the very beginning of a PERL script we change this parameter to enable a permanent TELNET session (never expire) and reset this parameter to the default value at the end of the script.

  Commands:

  ```
  Router-1# line vty 0 4
  Router-1(line)#exec time 0 0
  ```

PERL script to measure the time for write or erase one command in CISCO router, working with only 1 router.

```
#! PERL -w
use strict; #enforce some good programming rules
use Net::TELNET::Cisco;

#Router 1 - Eth0/0
my $router = '131.120.8.158';

#Load TELNET-CISCO module
my $S1 = Net::Telnet::Cisco->new(Host => $router);

#My variable to get the time of the Routers
my @output1;
my @output2;

#login to both router
$S1->login('monteiro','0ne4all');

#sh the timestamp-1
@output1 = $S1->cmd('sh clock');
$S1->cmd('config t');

#simple looping
for (1..1000) {
    $S1->cmd('ip rsvp sender 131.120.8.148 192.168.1.3 TCP 0 
0 192.168.1.1 Ethernet0/0 3750 7500');
    $S1->cmd('no ip rsvp sender 131.120.8.148 192.168.1.3 
TCP 0 0 192.168.1.1 Ethernet0/0 3750 7500');
}#end for

#exit priviledge mode
$S1->cmd('exit');

#sh the timestamp-2
@output2 = $S1->cmd('sh clock');

print "Router1 - t-1: @output1\n";
print "Router1 - t-2: @output2\n";
```

```
#Disable and exit router
$S1->disable;
$S1->close;
```

PERL script to measure the time for write or erase one command in CISCO router, working with 2 routers.

```
#! PERL –w
use strict; #enforce some good programming rules
use Net::Telnet::Cisco;

#Router 1 - Eth0/0
my $router1 = '131.120.8.158';
my $router2 = '131.120.64.4';

#Load TELNET-CISCO module
my $S1 = Net::Telnet::Cisco->new(Host => $router1);
my $S2 = Net::Telnet::Cisco->new(Host => $router2);

#My variable to get the time of the Routers
my @output1;
my @output2;
my @output3;
my @output4;


#login to both router
$S1->login('monteiro','xxxxxx');
$S2->login('monteiro','xxxxxx');

#sh the timestamp-1
@output1 = $S1->cmd('sh clock');
$S1->cmd('config t');
@output3 = $S2->cmd('sh clock');
$S2->cmd('config t');

#simple looping
for (1..1000) {
    $S1->cmd('ip rsvp sender 131.120.8.148 192.168.1.3 TCP 0
0 192.168.1.1 Ethernet0/0 3750 7500');
    $S1->cmd('no ip rsvp sender 131.120.8.148 192.168.1.3
TCP 0 0 192.168.1.1 Ethernet0/0 3750 7500');
    $S2->cmd('ip rsvp sender 131.120.8.148 192.168.1.3 TCP 0
0 192.168.1.1 Ethernet0/0 3750 7500');
    $S2->cmd('no ip rsvp sender 131.120.8.148 192.168.1.3
TCP 0 0 192.168.1.1 Ethernet0/0 3750 7500');
}#end for

#exit priviledge mode
$S1->cmd('exit');
$S2->cmd('exit');

#sh the timestamp-2
@output2 = $S1->cmd('sh clock');
@output4 = $S2->cmd('sh clock');

print "Router1 - t-1: @output1\n";
print "Router1 - t-2: @output2\n";
print "Router2 - t-1: @output3\n";
print "Router2 - t-2: @output4\n";

#Disable and exit router
$S1->disable;
$S1->close;
$S2->disable;
$S2->close;
```